

Backbone discovery in traffic networks

Sanjay Chawla
University of Sydney
Sydney, Australia
sanjay.chawla@sydney.edu.au

Aristides Gionis
Aalto University and HIIT
Helsinki, Finland
aristides.gionis@aalto.fi

Kiran Garimella
Aalto University
Helsinki, Finland
kiran.garimella@aalto.fi

Dominic Tsang
University of Sydney
Sydney, Australia
dwktsang@yahoo.com

ABSTRACT

We introduce a new computational problem, the BACKBONE-DISCOVERY problem, which encapsulates both *functional* and *structural* aspects of network analysis. For example, while the topology of a typical road network has been available for a long time (e.g., through paper maps), it is only recently that fine-granularity usage information about the network (like GPS traces), is being collected and is readily accessible. By combining functional and structural information, the solution of BACKBONE-DISCOVERY provides an efficient way to explore and understand usage patterns of networks and aid in design and decision making. To address the BACKBONE-DISCOVERY problem, we propose two algorithms that make use of the concepts of edge centrality. Our results indicate that it is possible to construct very sparse backbones that summarize the network activity very accurately. We also demonstrate that the usage of edge-centrality measures helps produce backbones of higher quality.

1. INTRODUCTION

Networks are routinely used to model interactions between entities, as well as flow of commodities between discrete nodes and locations. A commodity here may correspond to travelers, vehicles, or network packets. As data becomes readily available, detailed information is recorded not only about the structure of the underlying network, but also about the traffic that goes through it. Although in recent years there has been an increasing amount of work to analyze networks with respect to the observed activity, the majority of the graph-mining techniques focus on analyzing only the structural properties of the underlying networks. As more information becomes available with respect to the *function* of networks, standard approaches need to be augmented, and new problem formulations to be studied, in order to cater for the peculiarities of the new domain.

In this paper we study the problem of discovering the

backbone of traffic networks. We consider a setting in which we are given the topology of a network $G = (V, E)$ and a log $\mathcal{L} = \{(s_i, t_i, w_i)\}$, indicating an amount of traffic w_i that incurs between source s_i and destination t_i . We are also given a budget L that accounts for a total amount of edges available, potentially weighted by their cost. The goal is to discover a sparse subnetwork R of G , of cost at most L , which summarizes as well as possible the recorded traffic \mathcal{L} .

The problem we study has applications in diverse domains. From an *exploratory data analysis* perspective, the objective is to obtain a concise summarization of the data. For example in the case of Figure 1, this may correspond to extracting a very sparse subnetwork that represents most accurately the traffic pathways in the network. From a *network-design* perspective, given some limited budget, the goal is to find what are the most important edges to upgrade or to keep better maintained in order to minimize the total traffic disruptions.

The problem we consider is related to both the *k-spanner* and the *Steiner-forest* problem, both of which have been extensively studied in theoretical computer science. However, our problem formulation will have elements which are substantially distinct from both these problems.

In the *k-spanner* problem [9] the goal is to find a minimum-cost subnetwork R of G , such that for *each pair* of nodes u and v , the shortest path between u and v on R is at most k times longer than the shortest path between u and v on G . On the other hand, in the *Steiner-forest* problem [21] we are given a set of pairs of terminals $\{(s_i, t_i)\}$ and the goal is to find a minimum-cost forest on which each source s_i is connected to the corresponding destination t_i .

To understand the differences of the proposed backbone-discovery problem with the above two problems, consider the example shown in Figure 2. In this example, there are four group of nodes: group A consists of n nodes, a_1, \dots, a_n , group B consists of n nodes, b_1, \dots, b_n , group C consists of 2 nodes, c_1 and c_2 , and group D consists of m nodes, d_1, \dots, d_m . Consider that m is smaller than n , and thus much smaller than n^2 . All edges shown in the figure have cost 1, except the edges between c_1 and c_2 , which has cost 2. Assume that there is one unit of traffic between each a_i and each b_j , for $i, j = 1, \dots, n$, resulting to n^2 source-destination pairs (the majority of the traffic), and one unit of traffic between d_i and d_{i+1} , for $i = 1, \dots, m - 1$, resulting to $m - 1$ source-destination pairs (some additional marginal traffic). The example abstracts a layout that can be found in a number of cities: a few busy centers (commercial, residential,

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Under review

Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...\$15.00.

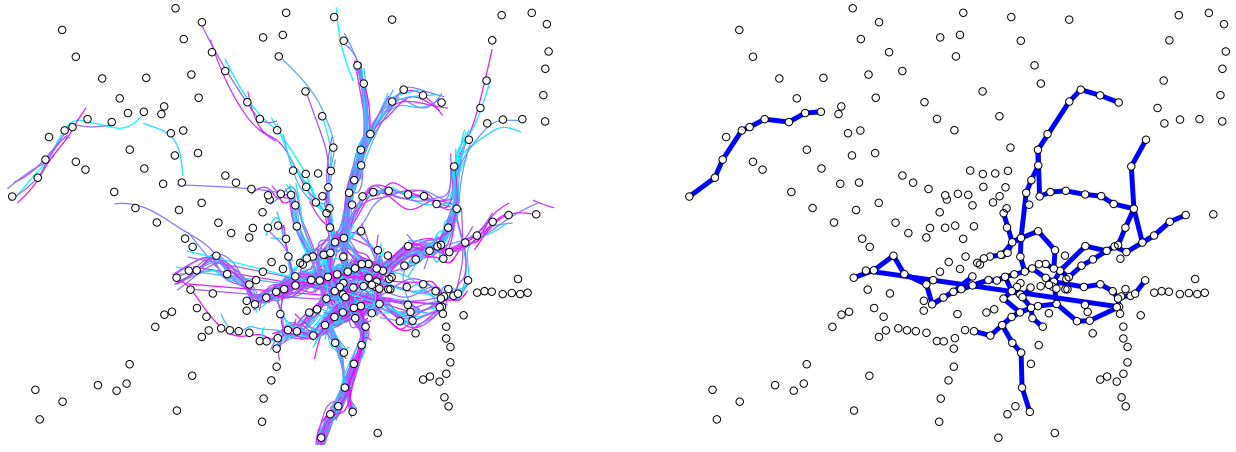
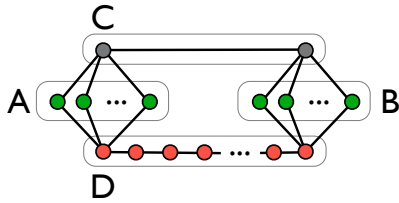
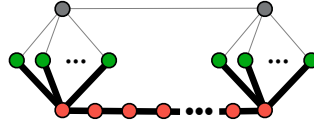


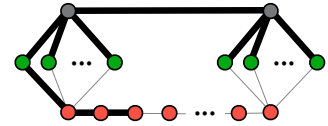
Figure 1: The figure on the left shows trips in the London tube, and the figure on the right shows the corresponding backbone, as discovered by our algorithm. The input data contains only source–destination pairs and for visualization purposes, a B-spline was interpolated along the shortest path between each such pair.



(a) A traffic network. We consider a unit of traffic from each node in A to each node in B , and from each node in D to its right neighbor.



(b) Shown with thick edges is an optimal Steiner forest for certain cost C .



(c) Shown with thick edges is a backbone of cost at most C that captures the traffic in the network better than the optimal Steiner forest.

Figure 2: The optimum Steiner tree solution may not be the best solution in our framework.

entertainment, etc.) with some heavily-used links connecting them, and some peripheral ways around, that also serve some additional traffic. For example in Sydney, Australia, the heavily-used and toll-restricted Harbor Bridge connects northern parts of the city with the business center but there is a longer and toll-free inland way which is available but sparingly used.

Careful inspection of the above example can help us gain a number of insights:

1. As opposed to the k -spanner problem, we do not need to guarantee short paths for all pairs of nodes, but only for those in our traffic log.
2. Since a budget is given as input, it may not be possible to guarantee connectivity for all pairs in the traffic log. This is compatible with our intended applications (e.g., discovery of important links, visualization of the network, link upgrade, etc.). Thus, we need a way to decide which pairs to leave disconnected. Neither the k -spanner nor the Steiner-forest problems provision for disconnected pairs. As a matter of fact, in contrast to the Steiner-forest formulation, it is conceivable that the optimal backbone may even contain cycles while leaving pairs disconnected.
3. Certain edges may be essential parts of the backbone,

even though other problem formulations can leave them out due to cost considerations. For example, while the edges that connects the nodes of set C is a very important edge for the overall traffic, the optimal Steiner-forest solution, shown in Figure 2b, prefers the long path along the nodes of the set D .

In this paper we formulate the backbone-discovery problem in a way that it addresses all the above considerations. We use the notion of *stretch factor*, defined as a *weighted harmonic mean* over the source–destination pairs of the traffic log, which provides a principled objective to optimize connectivity while allowing to leave disconnected pairs, when there is no sufficient budget.

To obtain low-cost solutions for the backbone-discovery problem, we propose two different algorithms based on greedy and primal-dual approaches, respectively. Both algorithms are enhanced with notions of edge centrality, such as *edge-betweenness centrality* and *commute-time centrality*.

The intuition for incorporating edge centrality measures into the problem is as follows. An algorithm to solve the Steiner-forest problem will try and minimize the sum of cost of edges selected as long as the set of terminal pairs $\{(s_i, t_i)\}$ are connected and is agnostic to minimizing stretch factor. However, if the edge costs are weighted with *edge-betweenness*, then there will be a bias to include edges through

which many of the shortest paths between terminals pass through. This will potentially help reduce the stretch factor. Similarly the use of *commute-time centrality* will relax the shortest path requirement and replace it with average path length and thus provide more generality. For example, consider the example in Figure 2c. By including *edge-betweenness* into the problem formulation, we can adapt a primal-dual algorithm for the *Steiner-forest* problem which will select the high edge-betweenness edge in component C as part of the solution. As already noted, the optimal solution of the standard cost-based *Steiner forest* will not include the edge in component C .

The rest of the paper is organized as follows. In Section 2, we introduce the notation and rigorously define the BACKBONEDISCOVERY problem. In Section 3 we survey related work and distinguish our problem formulation with other relevant approaches. Section 4 introduces two new algorithms based on greedy and primal-dual approaches. In Section 5 we overview network centrality measures and explain how they can be efficiently integrated into our solution framework. A detailed experimental evaluation, results and discussion is described in Section 5. We conclude in Section 6 with a summary and potential directions for future work.

2. PROBLEM DEFINITION

Let $G = (V, E)$ be a network, with $|V| = n$ and $|E| = m$. For each edge $e \in E$ there is a cost $c(e)$. Additionally, we assume that we are given a traffic log \mathcal{L} , specified as a set of triples (s_i, t_i, w_i) , with $s_i, t_i \in V$, and with $i = 1, \dots, k$. A triple (s_i, t_i, w_i) indicates the fact that w_i units of traffic have been recorded between nodes s_i and t_i .

We aim at discovering the *backbone* of traffic networks. A backbone R is a subset of the edges of the network G , that is, $R \subseteq E$. Intuitively, the backbone provides a good summarization for the whole traffic in \mathcal{L} . In particular, we require that if the available traffic had used only edges in the backbone R , it should have been almost as efficient as using all the edges in the network. We formalize this intuition below.

Given two nodes $s, t \in V$ and a subset of edges $A \subseteq E$, we consider the shortest path $d_A(s, t)$ from s to t that uses only edges in the set A . In this shortest-path definition, edges are counted according to their cost c . If there is no path from s to t using only edges in A , we define $d_A(s, t) = \infty$. Consequently, $d_E(s, t)$ is the shortest path from s to t using all the edges in the network, and $d_R(s, t)$ is the shortest path from s to t using only edges in the backbone R .

To measure the quality of a backbone R , with respect to some traffic log $\mathcal{L} = \{(s_i, t_i, w_i)\}$ we use the concept of *stretch factor*. Intuitively, we want to consider shortest paths from s_i to t_i , and evaluate how much longer are those paths on the backbone R , than on the original network.

In order to aggregate shortest-path information for all source–destination pairs in our log in a meaningful way, we need to address two issues. The first issue is that not all source–destination pairs have the same volume in the traffic log. This can be easily addressed by weighting the contribution of each pair (s_i, t_i) by its corresponding volume w_i .

The second issue is that since we aim at discovering very sparse backbones, many source–destination pairs in the log could be disconnected in the backbone. To address this problem we aggregate shortest-path distances using the *harmonic mean*. This idea, which has been proposed by Mar-

chiori and Latora [12] and has also been used by Boldi and Vigna [1] in measuring centrality in networks, provides a very clean way to deal with infinite distances. Indeed, if a source–destination pair is not connected, their distance is infinity, so the harmonic mean accounts for this by just adding a zero term in the summation. Using the arithmetic mean is problematic, as we would need to add an infinite term with other finite numbers.

Overall, given a set of edges $A \subseteq E$, we measure the connectivity of the traffic log $\mathcal{L} = \{(s_i, t_i, w_i)\}$ by

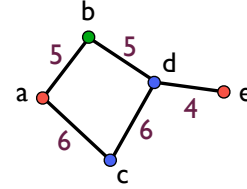
$$H_{\mathcal{L}}(A) = \left(\frac{1}{k} \sum_{i=1}^k \frac{w_i}{d_A(s_i, t_i)} \right)^{-1}.$$

The *stretch factor* of a backbone R is then defined as

$$\lambda_{\mathcal{L}}(R) = \frac{H_{\mathcal{L}}(R)}{H_{\mathcal{L}}(E)}.$$

The stretch factor is always greater or equal than 1. The closer it is to 1, the better the connectivity that it offers to the traffic log \mathcal{L} . The definition of stretch factor is analogous to that used in the setting of spanner graphs.

Example. Consider the network G in the figure below. Assume the log \mathcal{L} is $\{(a, e, 10), (c, d, 12)\}$. Furthermore assume we are given a budget $L = 18$ to build the backbone. The shortest path from a to e is $a \rightarrow b \rightarrow d \rightarrow e$. Thus $d_E(a, e) = 14$. Similarly the shortest path from c to d is just $c \rightarrow d$ and thus $d_E(c, e) = 6$.



If we take the two shortest paths as the backbone then the cost is 20 which is above the budget. However, if the backbone R consists of the edges $\{(a, c), (c, d), (d, e)\}$, then both (a, e) and (c, d) remain connected on the backbone and the resulting cost is 16, which is under budget. Now $d_R(a, e) = 16$ and $d_R(c, d) = 6$. The resulting stretch factor is

$$\lambda_{\mathcal{L}}(R) = \frac{\frac{10}{14} + \frac{12}{6}}{\frac{10}{16} + \frac{12}{6}} = 1.034$$

Now suppose the budget L is further reduced to 14. One possible backbone R is $\{(a, b), (b, d), (d, e)\}$. Notice that $d_R(c, d) = \infty$. The stretch factor is $\lambda_{\mathcal{L}}(R) = 4.34$. The other option is to choose R as $\{(c, d)\}$. Now $d_R(a, e) = \infty$. Now even though the full budget is not consumed the stretch factor is reduced to $\lambda_{\mathcal{L}}(R) = 1.38$.

The example suggests that a potentially complex and non-linear relationship exists between the network topology, the travel log and a budget to build the backbone. \square

We are now ready to formally define the problem of backbone discovery.

PROBLEM 1 (BACKBONEDISCOVERY). Consider a network $G = (V, E)$ and a traffic log $\mathcal{L} = \{(s_i, t_i, w_i)\}$. Consider also a cost budget L . The goal is to find a backbone network $R \subseteq E$ of total cost L that minimizes the average stretch factor $\lambda_{\mathcal{L}}(R)$ or report that no such solution exists.

As one may suspect, BACKBONEDISCOVERY is an **NP**-hard problem.

LEMMA 1. *The BACKBONEDISCOVERY problem, defined in Problem 1, is **NP**-hard.*

PROOF (SKETCH). We obtain a reduction from the SETCOVER problem: given a ground set $U = \{u_1, \dots, u_n\}$, a collection $\mathcal{S} = \{S_1, \dots, S_m\}$ of subsets of U , and an integer k , determine whether there are k sets in \mathcal{S} that cover all the elements of U .

Given an instance of the SETCOVER problem we form an instance of the BACKBONEDISCOVERY problem as follows. We create one node u_i for each $u_i \in U$ and one node v_j for each $S_j \in \mathcal{S}$. We also create a special node z . We create an edge (u_i, v_j) if and only if $u_i \in S_j$ and we assign to this edge cost 0. We also create an edge (v_j, z) for all $S_j \in \mathcal{S}$ and we assign to this edge cost 1. As far as the traffic log is concerned, we set $\mathcal{L} = \{(u_i, z, 1) \mid u_i \in U\}$, that is, we pair each $u_i \in U$ with the special node z with volume 1. Finally we set the budget $L = k$. It is not difficult to see that the instance of the BACKBONEDISCOVERY problem constructed in this way has a solution with stretch factor 1 if and only if the given instance of the SETCOVER problem has a feasible solution. \square

3. RELATED WORK

As noted above BACKBONEDISCOVERY is related to both the k -spanner and the Steiner-forest problem [9, 15, 21].

Given a network $G = (V, E)$, the k -spanner problem asks to find a subnetwork $H = (V, E')$ where the original distances between every pair of nodes are preserved within a factor k . Note that the node set of subgraph G and H are identical. The decision version of k -spanner problem is known to be **NP**-complete [15]. The problem has been extensively studied in the computer science theory community and several heuristic and approximation results have been proposed. For example, a very popular greedy heuristic for networks embedded in a metric space consists of the following steps: (i) Start with the nodes V of the original graph; (ii) Form a complete network $G' = (V, C)$; (iii) Sort the edges in C in increasing order; and (iv) Process each edge $e = (u, v)$ in the sorted order and issue a shortest-path query on the original network. If the length of the shortest path is greater than $k|u, v|$ then add the edge to H . It is known that the time complexity of the greedy algorithm is $\mathcal{O}(n^2)$. However, this greedy algorithm does not extend to the case where the objective is to only preserve the stretch factor for a subset of terminals, $\{(s_i, t_i)\}$. The focus on terminals connects the BACKBONEDISCOVERY problem to the Steiner-forest problem.

The input of Steiner-forest problem consists of a weighted undirected network $G = (V, E, w)$ and k pairs of nodes (s_i, t_i) . The objective is to find a subset of edges of E of minimum cost such that every (s_i, t_i) pair is connected. Again, the problem is **NP**-hard [21]. However, as we discussed in the introduction, the objective of Steiner-forest problem is not designed to preserve the stretch factor of the paths between the k terminals.

A major enhancement in our work is to integrate concepts related to the structure of the network (like edge centrality) into the problem we study. For example, instead of using plain distances as edge costs we incorporate edge betweenness [3, 10, 16] and commute time [6] to capture the central-

ity of an edge. As we will, this leads to finding solutions of better quality.

There has been some work in social network research to extract a subgraph from larger subgraphs subject to constraints. For example, Du et al. propose an algorithm, called *sketcher* which when applied to a network “extracts its backbone from the intertwined connections among the vertices” [8]. However, the term backbone is not formally defined and ostensibly means the set of vertices with high centrality. In another paper, Ruan et al. have formalized backbone discovery as an energy minimization problem consisting of two components [18]. The first component is based on shortest paths and the second on, what the authors term as the path recognition cost: an information-theoretic measure inspired from coding theory. Both of the above papers do not consider the availability of a travel log and neither characterize the stretch factor of the backbone.

The BACKBONEDISCOVERY problem is also related to finding graph sparsifiers and simplifying graphs. For example, Toivonen et al. [20] as well as Zhou et al. [22], propose an approach based on pruning edges while keeping the quality of best paths between all pairs of nodes, where quality is defined on concepts such as shortest path or maximum flow. Misiolek and Chen [14] propose an algorithm which prune edges while maintaining the source-to-sink flow for each pair of nodes. Mathioudakis et al. [13] and Bonchi et al. [2] study the problem of discovering the backbone of a social network in the context of information propagation, which is a different type of activity than source-destination pairs, as considered here.

In the computer network research community, the notion of software defined networks (SDN), which in principle decouples the network control layer from the physical routers and switches, has attracted a lot of attention [5, 11]. SDN (for example through OpenFlow) will essentially allow network administrators to remotely control routing tables. The BACKBONEDISCOVERY problem can essentially be considered as an abstraction of the SDN problem. We will provide some qualitative validation in the Experiments section.

Other forms of network backbone-discovery have been explored in domains including biology, communication networks and the social sciences. The main focus again is on the trade-off between the level of network reduction and the amount of relevant information to be preserved either for visualization or community detection. For example, according to Serrano et al. the backbone of dominant connections in weighted networks with strong disorder is filtered based on the “local identification of the statistically relevant weight heterogeneities” [19]. In the work of Butenko et al. a heuristic algorithm for the minimum connected dominating subset of wireless networks was proposed [4]. In another variation, Newman introduced a method for community detection based on modularity matrix analysis [17]. Again, BACKBONEDISCOVERY is distinct in that it casts the problem in a precise optimization framework where the *functional* aspects of the network are captured in the requirement to maintain a low stretch while the structural requirements are captured in maintaining connectedness between important terminals.

4. ALGORITHMS

The algorithms we propose for the BACKBONEDISCOVERY problem are divided in two families. The first family consists of variants of a *greedy* heuristic that connects one-by-one the

source–destination pairs of the traffic log \mathcal{L} . The algorithms in the second family are based on the primal–dual scheme for the *Steiner forest* problem [21].

A distinguishing feature of the algorithms in both families is that they utilize a notion of *edge benefit*. In particular, we assume that for each edge $e \in E$ we have available a benefit measure $b(e)$. The higher is the measure $b(e)$ the more beneficial it is to include the edge e in the final solution. The benefit measure is computed using the traffic log \mathcal{L} and it takes into account the global structure of the network G .

As we discussed in the introduction, the more central an edge is with respect to a traffic log, the more beneficial it is to include it in the solution, as it can be used to serve many source–destination pairs. We therefore employ edge centrality as a benefit measure. We experiment with two centrality measures, *edge-betweenness centrality*, and *commute-time centrality*. Both measures are adapted to take into account the traffic log.

Both of our algorithm families exploit the concept of edge benefit $b(e)$ in a similar way. In particular, they rely on the notion of *effective distance* $\widehat{\ell}(e)$, defined as $\widehat{\ell}(e) = c(e)/b(e)$, where $c(e)$ is the cost of an edge $e \in E$. The idea is to consider as cost for an edge e its effective distance $\widehat{\ell}(e)$, rather than its actual distance $c(e)$. The intuition is that by dividing the cost of each edge by its benefit, we are biasing the algorithms towards selecting edges with high benefit.

We now present our algorithms in more detail.

4.1 The greedy algorithm

As discussed above, our algorithms take advantage of a benefit score $b(e)$ computed for every edge $e \in E$, and consider as distance for its edge its effective distance $\widehat{\ell}(e) = c(e)/b(e)$. The objective is to obtain a cost/benefit trade-off: edges with small cost and large benefit are favored to be included in the backbone.

In the description of the greedy that follows we assume that the effective distance $\widehat{\ell}(e)$ of each edge is given as input.

The algorithm works in an iterative fashion, maintaining and growing the backbone, starting from the empty set. In the i -th iteration the algorithm picks a source–destination pair (s_i, t_i) from the traffic log \mathcal{L} , and “serves” it. Serving a pair (s_i, t_i) means computing a shortest path p_i from s_i to t_i , and adding its edges in the current R , if they are not already there. For the shortest-path computation the algorithm uses the effective distances $\widehat{\ell}(e)$. When an edge is newly added to the backbone its cost is subtracted from the available budget. Here, the actual cost of the edge $c(e)$ (instead of the $\widehat{\ell}(e)$) is used. Also its effective distance is reset to zero, since it can be used for free in subsequent iterations of the algorithm. The source–destination pair that is chosen to be served in each iteration is the one that reduces the stretch factor the most at that iteration; and hence the greedy nature of the algorithm.

The algorithm proceeds until it exhausts all its budget or until the stretch factor becomes equal to 1 (which means that all pairs in the log are served via a shortest path). The pseudocode for the greedy algorithm is shown as Algorithm 1.

We are experimenting with three variants of this greedy scheme, depending on the benefit score we use (and consequently on the effective distance). These are the following:

Greedy: We use uniform benefit scores ($b(e) = 1$).

Algorithm 1 The greedy algorithm

Input: Network $G = (V, E)$, edge costs $c(e)$, benefit costs $b(e)$, cost budget L , traffic log $\mathcal{L} = \{(s_i, t_i, w_i)\}$
Output: A subset of edges $R \subseteq E$ of total cost $c(R) \leq L$ and small stretch factor $\lambda(R)$

```

1: for all  $e \in E$  do
2:    $\widehat{\ell}(e) \leftarrow c(e)/b(e)$ 
3:  $R \leftarrow \emptyset$ 
4:  $\lambda \leftarrow \infty$ 
5: while ( $L > 0$ ) and ( $\lambda > 1$ ) do
6:   for each  $(s_i, t_i, w_i) \in \mathcal{L}$  do
7:      $p_i \leftarrow \text{SHORTESTPATH}(s_i, t_i, G, \widehat{\ell})$ 
8:      $\lambda_i \leftarrow \text{STRETCHFACTOR}(R \cup p_i, G, \mathcal{L})$ 
9:    $p^* \leftarrow \min_i \{\lambda_i\}$  // the min of the above iteration
10:   $R' \leftarrow p^* \setminus R$  // edges to be newly added
11:  if  $c(R') > L$  then
12:    Return  $R$  // budget exhausted
13:   $R \leftarrow R \cup R'$  // add new edges in the backbone
14:   $\widehat{\ell}(R') \leftarrow 0$  // reset cost of newly added edges
15:   $L \leftarrow L - c(R')$  // decrease budget
16:   $\lambda \leftarrow \text{STRETCHFACTOR}(R, G, \mathcal{L})$  // update  $\lambda$ 
17: Return  $R$ 

```

GreedyEB: The benefit score of an edge is set equal to its *edge-betweenness centrality*.

GreedyCT: The benefit score of an edge is set equal to its *commute-time centrality*.

Running time. Assuming that the benefit scores $b(e)$ are given for all edges $e \in E$ the worst-case running time of the algorithm is as follows. First assume that the algorithm performs I iterations until it exhausts its budget. In each iteration we need to perform $\mathcal{O}(k^2)$ shortest-path computations, where we remind that k is the size of the traffic log \mathcal{L} . A shortest path computation is $\mathcal{O}(m + n \log n)$, and thus the overall complexity of the algorithm is $\mathcal{O}(Ik^2(m + n \log n))$. The number of iterations I depends on the available budget and in the worst case it can be as large as k . However, since we aim at finding sparse backbones, the number of iterations is significantly smaller, and in practice it can be considered a constant.

Optimizations. In order to improve the performance of the greedy algorithm we introduce a number of heuristics. Since the most costly part of this approach is the computation of shortest paths on the newly-formed network, we make sure that we compute the shortest path only when needed. Our optimizations consist of two parts:

1. As the backbone grows during the execution of the algorithm, we maintain the connected components that it creates in the network. Then, we do not need to compute shortest paths for all (s_i, t_i) pairs, for which s_i and t_i belong in different connected components; we know that their distance is ∞ . This optimization is effective at the early stages of the algorithm, when many terminals belong in different connected components.
2. When computing the decrease in the stretch factor due to a candidate shortest path to be added in the backbone, for pairs for which we have to recompute a shortest-path distance, we first compute an optimistic

lower bound, based on the shortest path on the whole network (which we compute once in a preprocessing step). If this optimistic lower bound is not better than the current best stretch factor then we can skip the computation of the shortest path on the backbone.

As shown in the empirical evaluation of our algorithms, depending on the dataset, these optimization heuristics lead to 20–35% improvement in performance.

4.2 Algorithm based on primal–dual

In our introduction, with the help of Figure 2, we argued that the BACKBONEDISCOVERY problem that we address in this paper has important differences with the k -SPANNER and STEINERFOREST problems. Yet, existing approximation algorithms for the STEINERFOREST problem, which are based on the primal–dual method, offer an intuitive paradigm to think about sparsification of networks and connectivity of terminals. Thus, our second family of algorithms is a variation of the primal–dual scheme, inspired by an approximation algorithm for the STEINERFOREST problem [21] and adapted to our setting.

Recall that in the STEINERFOREST problem the objective is to find a minimum-cost tree that connects all pairs of terminals (s_i, t_i) . The primal–dual scheme is described in, e.g., the book of Williamson and Shmoys [21], but for completeness and in order to describe our algorithm better, we give a brief overview.

Basic primal–dual scheme. Given a source–destination pair (s_i, t_i) , denote by \mathcal{S}_i the subsets of nodes separating s_i and t_i , that is, $\mathcal{S}_i = \{S \subseteq V \text{ such that } |S \cap \{s_i, t_i\}| = 1\}$. Also denote by $\delta(S)$ set of edges that have one endpoint in S and the other not in S . Let c_e denote the cost of the edge e , and let x_e be a binary variable indicating whether the edge e is included in a solution. The Integer Program associated to the STEINERFOREST problem is the following:

$$\begin{aligned} & \text{minimize} \quad \sum_{e \in E} c_e x_e \\ & \text{such that} \quad \sum_{e \in \delta(S)} x_e \geq 1 \quad \text{for all } S \in \mathcal{S}_i \text{ for some } i. \end{aligned}$$

By relaxing the integrality constraint to $x_e \geq 0$ we obtain the dual of the resulting linear program:

$$\begin{aligned} & \text{maximize} \quad \sum_{\exists i: S \in \mathcal{S}_i} y_S \\ & \text{such that} \quad \sum_{S|e \in \delta(S)} y_S \leq c_e \quad \text{for all } e \in E \\ & \text{with } y_S \geq 0 \quad \text{for all } S \in \mathcal{S}_i \text{ for some } i. \end{aligned}$$

Let \mathcal{C} be the set of all the connected components C such that $|C \cap \{s_i, t_i\}| = 1$. A solution set of edges F is initialized to the empty set. The dual variables y_C , for all $C \in \mathcal{C}$, are increased at the same rate until a dual inequality becomes tight for some edge $e \in \delta(C)$ for a set C , whose dual is increased. The edge e is added to the solution F and the process continues. Once a feasible solution F is obtained, i.e., all pairs (s_i, t_i) are connected in F , we go through the edges in the reverse of the order in which they were added and if an edge can be removed without affecting the feasibility of the solution, it is deleted.

At the beginning of the algorithm the set \mathcal{C} of connected components consists only of the $2k$ singleton sets $\{s_i\}$ and

Algorithm 2 The primal–dual algorithm

Input: Network $G = (V, E)$, edge costs $c(e)$, benefit costs $b(e)$, cost budget L , traffic log $\mathcal{L} = \{(s_i, t_i, w_i)\}$
Output: A subset of edges $R \subseteq E$ of total cost $c(R) \leq L$ and small stretch factor $\lambda(R)$

- 1: $y \leftarrow 0$
- 2: $F \leftarrow \emptyset$
- 3: **while** not all (s_i, t_i) pairs are connected in (V, F) **do**
- 4: let \mathcal{C} be the set of all connected components C of (V, F) such that $|C \cap \{s_i, t_i\}| = 1$ for some i
- 5: Increase y_C for all C in \mathcal{C} uniformly until for some $e \in \delta(C')$, $C' \in \mathcal{C}$ is $\hat{\ell}_e = \sum_{S: e \in \delta(S)} y_S$
- 6: $F \leftarrow F \cup \{e\}$
- 7: $L' \leftarrow \sum_{e \in F} c_e$
- 8: **while** $(L' > L)$ **do**
- 9: Let $e' \in F$ select greedily such that by removing it from F , the stretch factor will increase the least
- 10: $F \leftarrow F - \{e'\}$
- 11: **while** $(L' < L)$ **do**
- 12: Let $e' \notin F$ select from E greedily such that the stretch factor will decrease the most
- 13: $F \leftarrow F \cup \{e'\}$
- 14: $R \leftarrow F$
- 15: **Return** R

$\{t_i\}$, that is, $\mathcal{C} = \bigcup_{i=1}^k \{s_i\} \cup \bigcup_{i=1}^k \{t_i\}$. As the algorithm proceeds, those connected components grow in size by adding more nodes to them, and by merging existing connecting components.

Adaptation for BACKBONEDISCOVERY. We first use the basic primal–dual scheme of growing and merging connecting components to obtain a tree of cost L' in which all (s_i, t_i) pairs of the traffic log are connected. If the cost of the tree at that point is smaller than the available budget ($L' < L$), we can add more edges and reduce further the stretch factor. We do this greedily: we add edges one-by-one and in each step we select the one that decreases the stretch factor the most. On the other hand, if the cost of the tree found by the expansion phase is larger than the available budget ($L' > L$), we need to remove edges. Again we do this in a greedy fashion: we remove edges one-by-one and in each step we select the one that increases the stretch factor the least. As with the greedy, when computing the total cost of the solution and comparing with the total budget L , the actual costs $c(e)$ are used. Pseudocode for the approach is given as Algorithm 2.

Algorithm variants. Similar to the greedy algorithm, the primal–dual algorithm first computes edge benefits $b(e)$ and effective costs $\hat{\ell}(e) = c(e)/b(e)$. It then applies the method described above, with edge costs $c_e = \hat{\ell}(e)$. Depending on the benefit scores used, uniform, edge-betweenness centrality, and commute-time centrality, we have three variants of the algorithm: PD, PDEB, and PDCT.

Running time: Again we assume that the benefit scores $b(e)$ are given for all edges $e \in E$. We can efficiently implement the primal–dual algorithm by using the *union-find* data structure to update connected components in roughly linear time. Merging two connected components requires changing the dual variables for up to m edges, and the algorithm merges two connected components at most m times.

Changing the dual variable for an edge can be done with a priority queue, requiring $\mathcal{O}(\log m)$ time. Thus, the overall running time of steps 1–7 in Algorithm 2 is $\mathcal{O}(m^2 \log m)$. After obtaining a Steiner forest, we have to add or remove edges greedily in steps 8–13 in Algorithm 2 to minimize the stretch factor. In each iteration in this step we need to compute the shortest paths $\mathcal{O}(km)$ times to select the best edge. Assuming that the total number of iterations is I , the total running time in the second part of the algorithm is $\mathcal{O}(Ikm(m + n \log n))$. As with the greedy, the number of iterations (I) can be considered a constant in practice.

4.3 Edge-betweenness centrality

As we already discussed in the previous sections, both our algorithmic schemes, greedy and primal–dual, use edge centrality measures for benefit scores $b(e)$. In this section we discuss the first centrality measure, edge betweenness, and in particular show how we adapt the measure to take into account the traffic log \mathcal{L} . In the next section we discuss our second measure, commute-time centrality.

We first recall the standard definition of edge-betweenness. Given a network $G = (V, E)$, we define $V_2 = \binom{V}{2}$ to be the set of all pairs of nodes of G . Given a pair of nodes $(s, t) \in V_2$ and an edge $e \in E$, we define by $\sigma_{s,t}$ the total number of shortest paths from s to t , and by $\sigma_{s,t}(e)$ the total number of shortest paths from s to t that pass through edge e .

The standard definition of edge-betweenness centrality $B(e)$ of edge e is the following:

$$B(e) = \sum_{(s,t) \in V_2} \frac{\sigma_{s,t}(e)}{\sigma_{s,t}}.$$

In other words, all pairs of nodes of the network contribute to the betweenness centrality of an edge, and they contribute with the same weight. In our problem setting we take into account the traffic log $\mathcal{L} = \{(s_i, t_i, w_i)\}$, and we define the edge-betweenness $B_{\mathcal{L}}(e)$ of an edge e with respect to log \mathcal{L} , as follows.

$$B_{\mathcal{L}}(e) = \sum_{(s,t,w) \in \mathcal{L}} w \frac{\sigma_{s,t}(e)}{\sigma_{s,t}}.$$

In this modified definition only the source–destination pairs of the log \mathcal{L} contribute to the centrality of the edge e , and the amount of contribution is proportional to the corresponding traffic.

As with the standard definition, the adapted edge-betweenness can be computed in time $\mathcal{O}(nm)$, where n and m is the number of nodes and edges in the network, respectively.

4.4 Centrality based on commute time

Edge-betweenness is based on shortest paths. In contrast, the commute time between two nodes s and t is defined as the expected *return* time for a random walk starting at node s provided that it will visit node t . Commute time is equivalently defined based on electric flow in networks. Given a network $G = (V, E)$ we associate an edge vector $c : E \rightarrow R$, to represent the conductance of each edge. Conductance and resistance are inversely related as $c(e) = 1/r(e)$ [6, 7].

Now suppose a unit of current is injected at a node s and extracted from a node t . We can define a vector f_{st} on V as

$$f_{st}(v) = \begin{cases} 1 & v = s \\ -1 & v = t \\ 0 & \text{otherwise.} \end{cases}$$

Ohm’s law states that $v(e) = i(e)r(e)$ where v and i are the voltage difference and current on the edge. Furthermore for a directed edge $e = (i, j)$, $v(e) = p(i) - p(j)$, where p denotes the absolute potential on each node. For a given vector f , we can use the Laplacian L of a network to capture the relationship $Lp = f$.

The Laplacian L of a network G is defined as

$$L_{ij} = \begin{cases} \sum_{e:i \in e} c(e) & i = j \\ -c(e) & e = (i, j) \\ 0 & \text{otherwise} \end{cases}$$

We can use the above definitions to relate with the travel log $\mathcal{L} = \{(s_i, t_i, w_i)_{i=1}^k$ and compute the benefit $b(e)$ of an arbitrary edge $e = (u, v)$.

Suppose for the pair (s_i, t_i) of source–destination pair we compute the potential vector p_i from the equation $Lp_i = f_i$. Now for each edge $e = (u, v)$ we define the *commute-time benefit* as

$$b(e) = c(e) \sum_{i=1}^k w_i (p_i(u) - p_i(v))$$

The definition of $b(e)$ is intuitive. The greater the potential difference on the edge e the larger the benefit.

5. EXPERIMENTAL EVALUATION

We carry out several experiments with the aim of contrasting our proposed algorithms and measures. We compare both families of algorithms, greedy and primal–dual, and their variants with the two centrality measures we consider. Overall, the algorithms we experiment with are Greedy, Greedy-EB, and GreedyCT, described in Section 4.3, and PD, PDEB, and PDCT, described in Section 4.4.

Baseline. In order to obtain better intuition for the performance of our methods we define a simple baseline approach where edges are added to the backbone one by one, in decreasing order of their effective distances $\hat{\ell}(e) = c(e)/b(e)$, where here $b(e)$ is edge-betweenness; among other simple baselines we tried, such as considering commute time or adding edges only in order of their centrality score, this was the best performing baseline.

5.1 Datasets

We use two types of datasets: real and semi-synthetic.

London Tube dataset. Our first dataset is obtained from the London Subway (Tube). The Tube network consists of subway stations and the links between them.¹ We estimate the cost of the edges of the network by obtaining the latitude and longitude of the stations and measuring the geographic distance between the stations. This is a proxy to more accurate distance functions that one would like to use, such as rail distance or travel time, but those are not available to us. The final network consists of 316 nodes and 724 links.

We also obtain a traffic log \mathcal{L} with the usage of the Tube from the Oyster card system.² This data consists of aggregate trips made by passengers between pairs of stations

¹<http://research.cs.queensu.ca/~daver/235/C1352963146/E20070302133910/Media/LondonUnderground.txt>

²<http://www.tfl.gov.uk/businessandpartners/syndication/16492.aspx>

during a one-month period (Nov–Dec 2009). We filter out source–destination pairs with traffic less than 100 and we remove bi-directional pairs by selecting one of them at random and summing up their traffic. We are left with 455 source–destination pair entries. Some manual cleaning was performed in order to match the network and the traffic log as they were obtained from different sources.

Semi-synthetic dataset. To study the behavior of the algorithm under different scenarios and traffic distributions, we consider *semi-synthetic* datasets. These datasets are generated using a *real* road network and synthetically generated traffic logs. For the road network we consider the UK road network.³ Each node of a network corresponds to a road intersection and each link corresponds to a road segment. Distances between nodes (edge cost) are computed again as the geographic distances between the junctions. We clean up the data and, for simplicity, use only the largest connected component. The final network we use consists of 8 341 nodes and 13 926 links.

We then generate synthetic traffic logs \mathcal{L} simulating different scenarios, where the links or the nodes have different importance. The traffic log in this case consists of pairs of junctions (s - t pairs) and the amount of traffic (workload) between them. We generate traffic-log datasets according to four different distributions: (i) power-law workload, power-law s - t pairs; (ii) power-law workload, uniformly random s - t pairs; (iii) uniformly random workload, power-law s - t pairs; and (iv) uniformly random workload, uniformly random s - t pairs. To generate power-law s - t pairs, we first generate a power-law distribution containing the nodes and then randomly sample from that distribution.

Abilene dataset. For a qualitative analysis we also consider a *real* dataset consisting of a sample of the network traffic extracted from the Internet2 backbone.⁴ This dataset allows us to visually validate the results of the algorithms. Known as *Abilene*, this dataset has been widely used in the communication-network research community, and consists of network traffic between major universities in the continental US. The network consists of twelve nodes and 15 high-capacity links. Associated with each physical link, we also have capacity of the link which will serve as a proxy for the cost of the link — high capacity links are more expensive. We obtain traffic logs from 2003 between 132 (144 – 12) pairs of distinct nodes.

5.2 Quantitative results

We now present the results of our experimental evaluation, where we compare our methods on the different datasets.

Budget vs. stretch factor. We examine the trade-off between budget and stretch factor for all our algorithms. Each algorithm is evaluated on the London Tube real dataset and the UK-Road semi-synthetic datasets. We evaluate all variants of the greedy scheme and the primal–dual using the two edge-centrality measures, edge betweenness and commute time, described previously.

Figure 3(a-d) shows the trade-off between budget and stretch factor for the UK Road datasets and Figure 5 shows the same trade-off for the London Tube dataset. In all figures the budget used by the algorithms, shown in the x -axis,

is expressed as a percentage of the total cost of all the edges in the network.

We would like to highlight the following key takeaways.

1. Both families of algorithms, greedy and primal–dual, perform much better than the baseline. Note that baseline is not included in Figure 3(a,c) because the edges in the baseline are added one-by-one and for a large interval of the cost, the stretch factor was very large or sometimes even infinity.
2. The backbones discovered by our algorithms are sparse and summarize well the given traffic. In all cases, with about 8% of the edge cost in the network it is possible to summarize the traffic with stretch factor close to 1. In some cases, even less budget is needed to reach at a low stretch-factor value.
3. For both algorithms, the stretch factor decreases as the budget increases. This is expected as increase in budget implies the ability to include more edges in the backbone and thus leading to a decrease in the stretch factor.
4. The greedy scheme almost always performs better than primal–dual. This could be due to the fact that the main objective of greedy is to reduce the stretch factor, where as for primal–dual the main objective is to obtain a more connected backbone. In Figure 3(d), the difference between the greedy and the primal–dual family is most stark: the greedy stretch factor is smaller by nearly 300% compared to primal–dual, in some cases. The one exception is the London Tube dataset, shown in Figure 3(e), where the primal–dual family has a substantially lower stretch factor in the low-cost regime before all algorithms converge.
5. Incorporating edge betweenness and commute time in the algorithms improves the performance, there is at least a 50% increase in stretch factor all cases.

Budget and size of backbone. The greedy algorithm operates by connecting s - t pairs in each iteration and reducing stretch factor. On the other hand, the primal–dual algorithm works by adding or removing edges from the Steiner forest. One would expect that for a given cost, primal–dual will keep a larger percentage of s - t pairs connected than greedy. However, as we can see in Figure 4, we did not observe this behavior: greedy seems to be able to connect the same fraction of s - t pairs, for the same budget. This can be partially explained by the fact that the standard reverse deletion step of the primal–dual algorithm has been replaced by a greedy step to reduce stretch factor. A deeper analysis of the behavior of the primal–dual algorithm for the problem is warranted to draw firmer conclusions.

Performance improvement with heuristics. As described in Section 4.1, our greedy algorithm scales as $\mathcal{O}(k^2(n + m \log m))$. Since this might not scale well for large networks, we proposed two heuristics to improve the performance of the greedy algorithm. Figure 6 shows a comparison of the time taken to run our algorithm with and without the heuristics. We observe that in most cases, there is at least around 30% improvement in runtime.

³<http://www.dft.gov.uk/traffic-counts/download.php>

⁴<http://www.internet2.edu>

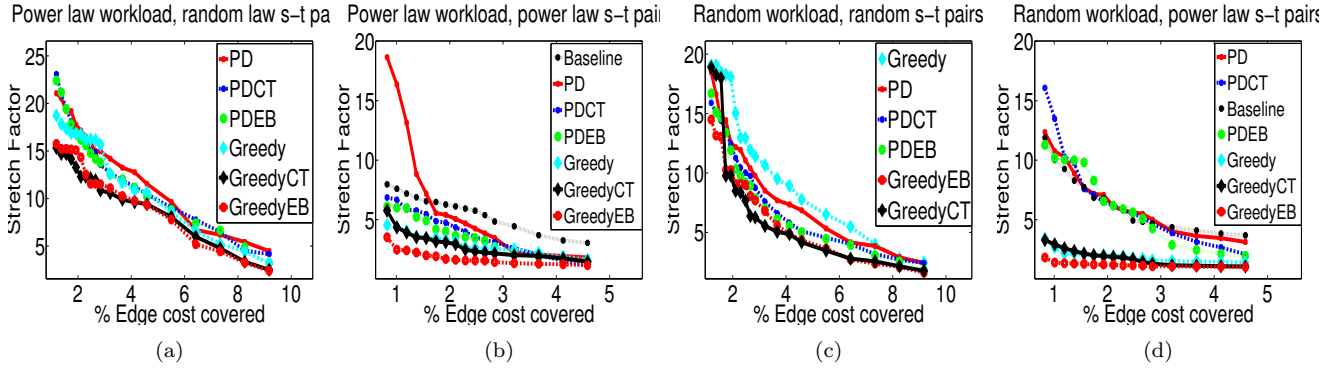


Figure 3: The relationship between the stretch factor and budget (expressed as the percentage of the total edge cost used) for the different algorithms on the UK Road dataset. Note that in 3a and 3c, the baseline is almost always infinity since the graph is not connected for a large interval of the cost values shown in the figure.

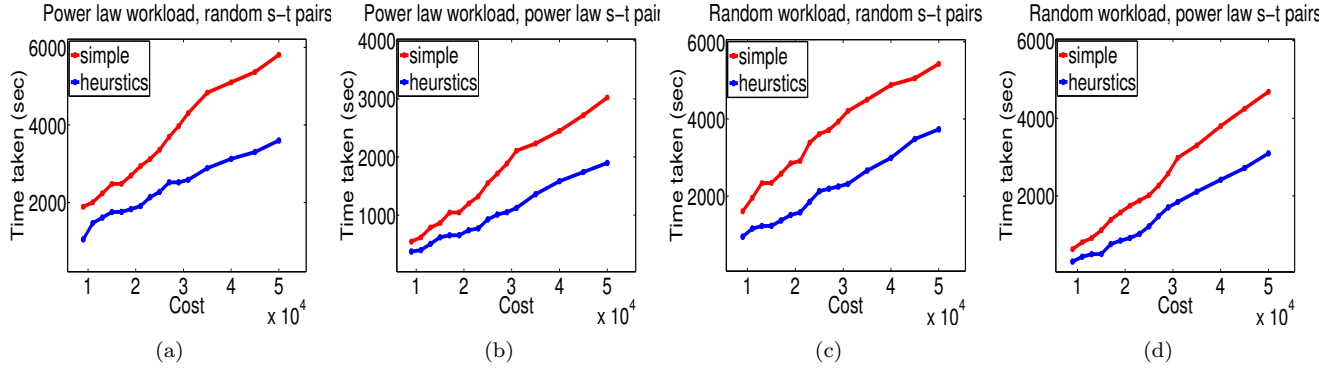


Figure 6: Comparison of the time taken to run GreedyEB with and without the heuristics we proposed for the UK Road dataset. The proposed heuristics show around 30% improvement in the runtime.

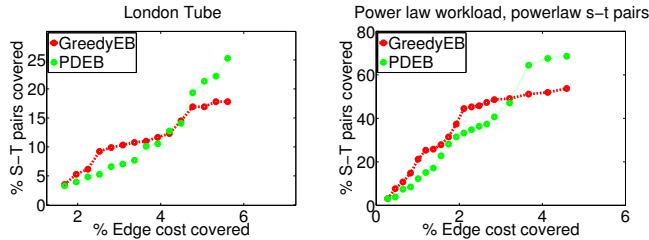


Figure 4: The relationship between the number of $s-t$ pairs added to the back bone and the budget for PDEB and GreedyEB on two datasets, the London tube, and the Powerlaw workload, powerlaw $s-t$ pair distribution of the UK Road data. Even though we plotted these only for PDEB and GreedyEB the results are similar for all other variants.

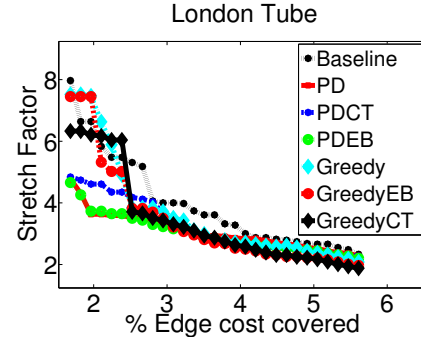


Figure 5: Relationship between stretch factor and budget on the London tube dataset.

5.3 Case study

We now carry out a qualitative analysis of the BACKBONEDISCOVERY problem on the Abilene dataset. The results of the application of PD and Greedy are shown in Figure 7. Note that two nodes in Atlanta have been merged.

The results provide preliminary evidence that the BACKBONEDISCOVERY problem can be tightly integrated with software defined networks (SDN), an increasingly important area in communication networks. The objective of SDN is to allow a software layer to control the routers and switches in

the physical layers based on the profile and shape of the traffic [5, 11]. This is precisely what the BACKBONEDISCOVERY solution is accomplishing in Figure 7. The design of data-driven logical networks will be an important operation implemented through an SDN. For example, in this particular case the BACKBONEDISCOVERY solution can help network designers answer the following question: “Given the expected traffic flow between nodes of a network and a budget L , how should routing tables be modified globally, in order to improve the overall aggregate network experience?”

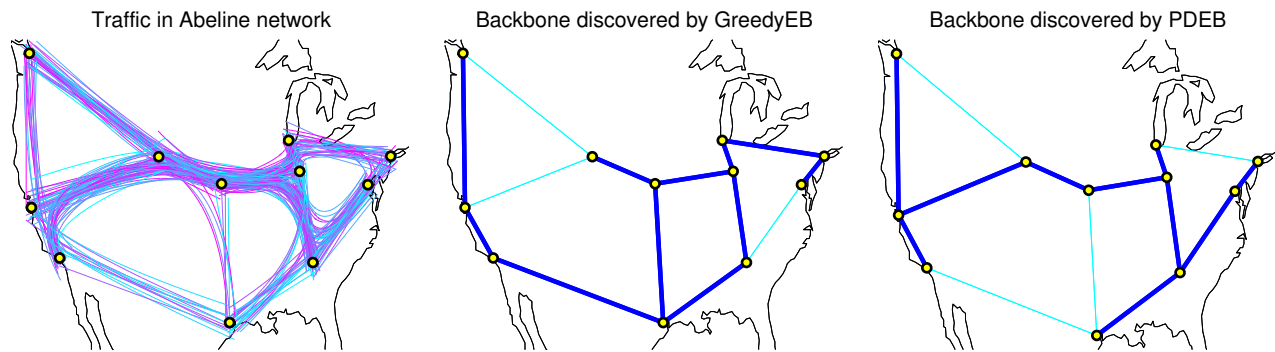


Figure 7: Qualitative analysis of the real Internet network. The figure on the left shows network traffic in the Abeline dataset, and the other two figures show the backbones discovered by the algorithms GreedyEB and PDEB, respectively. As with Figure 1 the traffic shown is not the actual one. The data contain only source–destination pairs and a curve was interpolated along the shortest path between each such pair.

6. CONCLUSIONS

In this paper we have introduced a new problem BACKBONEDISCOVERY to address a modern phenomenon: these days not only is the *structural* information of a network available but increasingly, highly granular *functional* information related to network usage is accessible. For example, the aggregate traffic usage of the London Subway (the Tube) between all stations, is available from a public website. The BACKBONEDISCOVERY problem allowed us to efficiently combine structural and functional information to obtain a highly sophisticated understanding of how the Tube is used (See Figure 1). From a computational perspective, the BACKBONEDISCOVERY problem has elements of both the k -spanner and the Steiner forest problem. This required the design of new algorithms to simultaneously maintain a low stretch factor and connectedness between important nodes of the network for a given budget cost. Future work includes more diverse applications of the problem introduced and potentially a deeper theoretical analysis of the algorithms proposed.

7. REFERENCES

- [1] P. Boldi and S. Vigna. Axioms for centrality. *CoRR*, abs/1308.2140, 2013.
- [2] F. Bonchi, G. De Francisci Morales, A. Gionis, and A. Ukkonen. Activity preserving graph simplification. *DMKD*, 27(3), 2013.
- [3] U. Brandes and C. Pich. Centrality estimation in large networks. *International Journal of Bifurcation and Chaos*, 17(7), 2007.
- [4] S. Butenko, X. Cheng, C. A. Oliveira, and P. M. Pardalos. A new heuristic for the minimum connected dominating set problem on ad hoc wireless networks. *Springer*, 3, 2004.
- [5] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. Gude, N. McKeown, and S. Shenker. Rethinking enterprise network control. *IEEE/ACM Trans. Netw.*, 17(4):1270–1283, 2009.
- [6] A. Chandra, P. Raghavan, W. Ruzzo, R. Smolensky, and P. Tiwari. The electrical resistance of a graph captures its commute and cover times. *CC*, 1996.
- [7] P. Doyle and J. L. Snell. Random walks and electric networks. *The Mathematical Association of America*, 1984.
- [8] N. Du, B. Wu, and B. Wang. Backbone discovery in social networks. *Web Intelligence*, 2007.
- [9] M. Elkin and D. Peleg. Approximating k -spanner problems for $k > 2$. In *IPCO*, 2001.
- [10] M. Girvan and M. E. J. Newman. Community structure in social and biological network. *PNAS*, 2002.
- [11] H. Kim and N. Feamster. Improving network management with software defined networking. *IEEE Communications Magazine*, 51(2):114–119, 2013.
- [12] M. Marchiori and V. Latora. Harmony in the small world. *Physica A: Statistical Mechanics and its Applications*, 285, 2000.
- [13] M. Mathioudakis, F. Bonchi, C. Castillo, A. Gionis, and A. Ukkonen. Sparsification of influence networks. In *KDD*, 2011.
- [14] E. Misiolek and D. Z. Chen. Two flow network simplification algorithms. *IPL*, 97, 2006.
- [15] G. Narasimhan and M. Smid. *Geometric Spanner Networks*. Cambridge University Press, 2007.
- [16] M. E. J. Newman. A measure of betweenness centrality based on random walks. *Social Networks*, 27(1), 2005.
- [17] M. E. J. Newman. Modularity and community structure in networks. *PNAS*, 103(23), 2006.
- [18] N. Ruan, R. Jin, G. Wang, and K. Huang. Network backbone discovery using edge clustering. *arXiv:1202.1842*, 2012.
- [19] M. Serrano and A. Vespignani. Extracting the multiscale backbone of complex weighted networks. *PNAS*, 106(16), 2009.
- [20] H. Toivonen, S. Mahler, and F. Zhou. A framework for path-oriented network simplification. In *IDA*, 2010.
- [21] D. Williamson and D. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.
- [22] F. Zhou, S. Mahler, and H. Toivonen. Network simplification with minimal loss of connectivity. In *IDA*, 2010.